Laboratory work nr. 2 part 1

Scebec Mihai, IS-211M

In this laboratory we are working with two different datasets, so I decided to divide the work into two different files. I will start with the imports, getting the dataset.

In [21]:
```
# Just to make things clear - we have 2 different datasets, so I'd like to have them in separate ipynb files
# imports
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.svm import SVC
from sklearn.metrics import classification_report, confusion_matrix, accuracy_score
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import GridSearchCV
import numpy as np
```

In [22]:
```
# get dataset
dataset = pd.read_csv('Social_Network_Ads.csv')
print(dataset.head(10))
```

```
   Age  EstimatedSalary  Purchased
0   19            19000          0
1   35            20000          0
2   26            43000          0
3   27            57000          0
4   19            76000          0
5   27            58000          0
6   27            84000          0
7   32           150000          1
8   25            33000          0
9   35            65000          0
```

In [23]:
```
# I got a wise advice to actually START using np arrays instead of dataframes.
X = np.array(dataset[['Age', 'EstimatedSalary']])
y = np.array(dataset['Purchased'])
```

In [24]:
```
# another advice I got is to train and fit the model before splitting it into train and test variables
standard_scaler = StandardScaler()
scaled_matrix = standard_scaler.fit_transform(X)
```

In [25]:
```
# split into tran and test data
X_train, X_test, y_train, y_test = train_test_split(scaled_matrix, y, test_size=0.2)
```

One wise friend of mine told me to scale dataset before splitting and working with it. He was absolutely right. This, together with working with numpy arrays instead of dataframes sped up the work greatly. Before I had freezing PC and lagging ipy kernel, now it takes seconds to process.

In [26]:
```
# create SVC, make a prediction
svc_classifier = SVC(kernel='linear', random_state = 0)
svc_classifier.fit(X_train, y_train)
svc_prediction = svc_classifier.predict(X_test)
svc_prediction
# well, not exactly the way to see the results
```

Out[26]:
```
array([0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0,
       1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1,
       0, 0, 0, 0, 0, 1, 0, 0, 1, 1, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0,
       0, 1, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0], dtype=int64)
```

In [27]:
```
# compare prediction with actual results
print(confusion_matrix(y_test,svc_prediction))
print(classification_report(y_test,svc_prediction))
# results differ from one predict to another, but precision in bigger than recall most of the times
# and in general looks good enough
#
# Also this is worth mentioning that without 'fit and transform' function before splitting data into
# train and test I was getting worse results than here
print(accuracy_score(y_test, svc_prediction))
# 0.9 accuracy seems surprisingly high, I got way worse results last attempt
```

```
[[50  4]
 [11 15]]
              precision    recall  f1-score   support

           0       0.82      0.93      0.87        54
           1       0.79      0.58      0.67        26

    accuracy                           0.81        80
   macro avg       0.80      0.75      0.77        80
weighted avg       0.81      0.81      0.80        80

0.8125
```

Here I see that from 80 values we are getting 15 of them wrong, this seems pretty ok-ish already, but, obviously, not perfect.

In [28]:
```
# now we will cross validate the result to see if we can reach more accuracy
scores = cross_val_score(svc_classifier, scaled_matrix, y, cv=10)
print(scores)
print(scores.mean())
# Ugh, it seems we were just very lucky
```

```
[0.675 0.675 0.975 0.95  1.    0.875 0.8   0.775 0.8   0.675]
0.82
```

Cross validation shows that what I got above is semi what I can expect to get from this model everytime, but, ofc, even cross validation gave me different results from try to try. Some of them were above .90

In [30]:
```
# next step is grid search
# set C parameters and kernel
param_grid = [
  {'C': [0.25, 0.5, 0.75, 1], 'kernel': ['linear','rbf']}
 ]

grid_search_result = GridSearchCV(svc_classifier, param_grid, scoring="accuracy", n_jobs= -2)
grid_search_result.fit(X_train, y_train)


print("Best parameters set found on development set: "+str(grid_search_result.best_params_))
print()
print("Grid scores on development set:")
print()
means = grid_search_result.cv_results_["mean_test_score"]
stds = grid_search_result.cv_results_["std_test_score"]
best_result = {}
mean_zero = 0
# zip to loop through several lists
for mean, std, params in zip(means, stds, grid_search_result.cv_results_["params"]):
    if(mean_zero < mean):
        mean_zero = mean
        best_result['mean'] = mean
        best_result['params'] = params
    print("%0.3f (+/-%0.03f) for %s" % (mean, std * 2, params))
print()
print('Best results found with %0.3f mean and parameters %s' % (best_result['mean'], best_result['params']))
# to be fair, all 3 rbf kernels showed the same result
print("classification report:")
print()
y_true, y_pred = y_test, grid_search_result.predict(X_test)
print(classification_report(y_true, y_pred))
```

```
Best parameters set found on development set: {'C': 0.5, 'kernel': 'rbf'}

Grid scores on development set:

0.834 (+/-0.081) for {'C': 0.25, 'kernel': 'linear'}
0.900 (+/-0.051) for {'C': 0.25, 'kernel': 'rbf'}
0.844 (+/-0.066) for {'C': 0.5, 'kernel': 'linear'}
0.909 (+/-0.046) for {'C': 0.5, 'kernel': 'rbf'}
0.844 (+/-0.079) for {'C': 0.75, 'kernel': 'linear'}
0.909 (+/-0.046) for {'C': 0.75, 'kernel': 'rbf'}
0.844 (+/-0.079) for {'C': 1, 'kernel': 'linear'}
0.909 (+/-0.046) for {'C': 1, 'kernel': 'rbf'}

Best results found with 0.909 mean and parameters {'C': 0.5, 'kernel': 'rbf'}
classification report:

              precision    recall  f1-score   support

           0       0.94      0.93      0.93        54
           1       0.85      0.88      0.87        26

    accuracy                           0.91        80
   macro avg       0.90      0.91      0.90        80
weighted avg       0.91      0.91      0.91        80
```

For given parameters the code found best fit, that is .9 accuracy, pretty good. Second csv files seems to have better dataset for training than the first one.

As a small conclusion pre-processing incoming dataset and then further tuning the regression parameters can increase accuracy of the model by at least 10%, which is not a bad result, given we started with 80%. But we can do more using more regression types.